Testing Stochastic Software using Pseudo-Oracles

Matthew Patrick
Department of Plant Sciences
University of Cambridge
United Kingdom
mtp33@cam.ac.uk

Andrew P. Craig
Department of Plant Sciences
University of Cambridge
United Kingdom
apc53@cam.ac.uk

Nik J. Cunniffe
Department of Plant Sciences
University of Cambridge
United Kingdom
njc1001@cam.ac.uk

Matthew Parry
Department of Mathematics
& Statistics
University of Otago
New Zealand
mparry@maths.otago.ac.nz

Christopher A. Gilligan
Department of Plant Sciences
University of Cambridge
United Kingdom
cag1@cam.ac.uk

ABSTRACT

Stochastic models can be difficult to test due to their complexity and randomness, yet their predictions are often used to make important decisions, so they need to be correct. We introduce a new search-based technique for testing implementations of stochastic models by maximising the differences between the implementation and a pseudo-oracle. Our technique reduces testing effort and enables discrepancies to be found that might otherwise be overlooked. We show the technique can identify differences challenging for humans to observe, and use it to help a new user understand implementation differences in a real model of a citrus disease (Huánglóngbìng) used to inform policy and research.

CCS Concepts

•Software and its engineering → Software creation and management; Software testing and debugging;

Keywords

computational models; testing; search-based optimisation

1. INTRODUCTION

Stochastic models are used to inform key decisions on a wide variety of topics, ranging from finance [4] and healthcare [21] through to epidemiology [6] and conflict [11], as well as in other important areas in industry and science. Over half of scientists develop more software now than they did 10 years ago [16]. In a recent survey [23], 70% of biological, mathematical and physical science researchers said they develop software as part of their job and 80% claimed it would be impossible to conduct their work without such software. Scientists need to have confidence in their models, because the predictions they make can have far-reaching consequences. For example, during the Space Shuttle Columbia incident, NASA dismissed the predictions of their model because it was thought to be over-conservative [40], but during descent the Shuttle disintegrated and the crew were killed.

Computational models can be challenging to work with because of their high levels of *essential* and *accidental* complexity [37]. Essential complexity occurs due to the need to represent the intricate details of biological, chemical or physical real-world systems. For example, in epidemiology, computational models are used to describe behaviour over a range of spatiotemporal scales. These models often involve complex interactions between biological species, with nonlinear dynamics. Accidental complexity arises from a lack of clarity in model implementations due to programming language restrictions and performance optimisations. Scientists are typically trained in their own field of research rather than in software engineering, so may develop programs that are disorganised and difficult to read [34].

One frequent source of essential complexity is stochasticity. Stochasticity is incorporated into a model to capture non-deterministic elements of the phenomenon being studied, or if there are unquantified sources of error to account for. Stochasticity introduces new difficulties into testing models, due to the inherent randomness of the output. For each set of inputs that are of interest, the output is unlikely to be unique. To test the model we must therefore consider the distributions produced over multiple executions.

This paper introduces a new technique to test stochastic computational models by applying search-based optimisation to the parameter values of multiple independent implementations (developed by different people), to maximise the differences in their output. Our technique is based on the concept of pseudo-oracles (also known as differential testing [31], dual coding [45] and N-version programming [22]). The discrepancies between implementations may initially be subtle, but our technique is able to find parameter values for which the differences are more obvious. The nature of these differences can then provide insight into the causes of the discrepancy. For example, if the model contains many variables but the difference is much more pronounced in one of those variables, the programmer can start their investigation by inspecting the lines of code dealing with that variable.

We demonstrate this technique on two case studies: (i) artificially introduced discrepancies in implementations of a simple epidemiological model, and (ii) differences between two real implementations of a more complex model, including the 'wrapper' code written to interface with the implementations by a scientist who was learning how to use them.

The paper is organised as follows: Section 2 provides some background concerning pseudo-oracles, Section 3 explains our technique and Section 4 introduces the models we apply it to; We describe our research questions and experiments in Sections 5 and 6; The results are analysed in Section 7, the threats to their validity in Section 8; We detail related work in Section 9 and present our conclusions in Section 10.

2. TESTING USING PSEUDO-ORACLES

Automated oracles are important for testing software thoroughly and efficiently [2]. Pseudo-oracles are used to test software for which no oracle is available, either because it is too difficult to determine what the correct output should be, or because an oracle is impossible to produce [45]. Pseudo-oracles compare the outputs of multiple independent implementations and check to see if there are any discrepancies.

Pseudo-oracles have been applied to a wide variety of software, such as compilers [31], access control systems [27] and refactoring engines [8]. For example, pseudo-oracles were used to test 9 commercial packages for seismic oil exploration [22]. These packages were found to produce significantly different results on the same input data, due to problems such as off-by-one errors. Predictions made from the packages were substantially different, such that people using them would come to different conclusions, potentially leading to $20 million oil wells being dug in the wrong place [22]. Other recent research into pseudo-oracles includes [5] and [46].

Finding a suitable pseudo-oracle may not necessarily be easy, as models are often developed for highly specialised purposes. However, two implementations do not have to be completely identical in intended function; as long as they can be parametrised/restricted to behave the same way, they can be treated as partial pseudo-oracles for each other. Our technique can be used to identify the differences between independent implementations of a model, so they can be addressed appropriately through pre/post-processing scripts.

NB: Scientists often use 'model' to refer interchangeably to both a theoretical model and the software in which that theoretical model is implemented. In this paper, we distinguish those two concepts: we use *model* to refer only to a theoretical (usually mathematical) model, and *implementation* to refer only to the software implementation.

3. SEARCHING FOR DIFFERENCES

The technique introduced in this paper applies search-based optimisation to improve the accuracy of a pseudo-oracle approach for identifying differences between multiple implementations of a stochastic model. Statistics are collected to characterise the distributions of outputs produced by each implementation (for a given set of input parameters) and search is used to move through the parameter space and maximise the difference between the distributions of each implementation. By making the differences more apparent, information is provided to the scientist (e.g., which model variable(s) are affected, and with what timing) that make it easier to identify the causes of any discrepancies.

3.1 Comparing implementation outputs

Quantifying the 'difference' between the outputs of stochastic implementations is non-trivial, and a suitable methodology must be chosen. To account for the inherent randomness in the output, we run each implementation a number of times. Maximising the difference (e.g., squared error) of the means of some output produced by these runs might be the simplest approach, but would ignore the potentially large variance due to stochasticity. It is also not feasible to compare every possible pair of outputs, as this would require too many comparisons. We have therefore chosen to collect a set of summary statistics to characterise the outputs of each implementation. Our algorithm then maximises the differences between those summary statistics.

Although our technique can be applied to test a wide range of software, we have chosen to focus on time series, because many stochastic models output information in this form. Time series can be characterised by fitting a parametrised distribution to their data, then using the distribution's parameters as summary statistics. However, this requires the distribution to be chosen in advance. We have taken a more general approach, recording certain key features from the time series [35]. The statistics we record from each time series are: (i) the area under the curve (AUC), (ii) the value of the peak and (iii) the time at which the peak occurs. For other types of models, e.g. financial models, choosing different statistics may improve algorithm performance.

We use the summary statistics (AUC, peak time, peak value) to characterise the time series output by each implementation. Running the implementations multiple times produces sets of these statistics. For each summary statistic we compare the outputs of our two implementations using the Kolmogorov-Smirnov test [29], a nonparametric test for the difference of two probability distributions. We use the resulting p-values as a measure of difference (note we are not performing hypothesis testing). The Kolmogorov-Smirnov test is preferable compared to other tests, such as those based on mean or median difference, because it takes into account the shape of the distributions (program output distributions may be different, even if they have the same mean). Finally, we take the lowest of these resulting p-values, and use this — denoted f — as the measure of difference between the outputs of the two implementations. We use f (see Equation 1) as the objective function for our search-based optimisation, described in the next section.

$$f = \min_{k \in \{1 \dots \kappa\}} \begin{pmatrix} ks(\text{peak\_value}_{k,1}, \text{peak\_value}_{k,2}) \\ ks(\text{peak\_time}_{k,1}, \text{peak\_time}_{k,2}) \\ ks(\text{AUC}_{k,1}, \text{AUC}_{k,2}) \end{pmatrix} \tag{1}$$

(κ is the number of time series, $ks(a, b)$ is the Kolmogorov-Smirnov test p-value for a and b, $\text{peak\_value}_{k,1}$ is the set of peak values recorded for all times series of type k (e.g., 'number of infected individuals') from implementation 1, and similarly for the other statistics.)

3.2 Search-based optimisation

We use search-based optimisation to find sets of input parameters that optimise (maximise) the difference between the outputs of the two implementations we are testing. The landscape of fitness values produced by Equation 1 is highly noisy due to stochasticity. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [18] has been shown to be effective on noisy landscapes [20]. In addition, this paper

Algorithm 1 Optimising model parameters θ

```
Input: $\mu_\theta$ (initial means for parameters), $\sigma$ (initial standard deviation for Gaussian adaptation), $\lambda$ (population size), $n$ (number of simulations for each candidate), $\kappa$ (number of time series), $M^1, M^2$ (model implementations)
repeat
    $cov \leftarrow$ identity matrix
    $f \leftarrow \{\}$
    for $i \in \{1 \dots \lambda\}$ do                                  # iterate over the population
        $\theta_i \leftarrow \mathcal{N}(\mu_\theta, \sigma^2 cov)$
        for $m \in \{1, 2\}$ do                                         # iterate over the implementations
            for $k \leftarrow \{1 \dots \kappa\}$ do
                $pv_k^m \leftarrow \{\}, pt_k^m \leftarrow \{\}, auc_k^m \leftarrow \{\}$
            end for
            for $j \in \{1 \dots n\}$ do
                $Y \leftarrow M^m(\theta_i)$                            # $M^m(\theta_i)$ is a stochastic simulation from the implementation
                for $k \leftarrow \{1 \dots \kappa\}$ do
                    $pv_k^m.append(max(Y_k))$
                    $pt_k^m.append(argmax(Y_k))$
                    $auc_k^m.append(AUC(Y_k))$
                end for
            end for
        end for
        $f.append(\min_{k \in \{1 \dots \kappa\}}(ks(pv_k^1, pv_k^2), ks(pt_k^1, pt_k^2), ks(auc_k^1, auc_k^2))$
    end for
    sort $\{\theta_i\}$ with respect to $\{f_i\}$
    update $\mu_\theta$, $\sigma$ and $cov$                               # using maximum likelihood estimation [18]
until stopping condition met
```

offers a proof of concept for our technique, which may be used on more complex software, with other forms of input. One attractive feature of CMA-ES is that it operates efficiently without the need for much parameter tuning. We use the latest CMA-ES libraries for Python [17] and R [42].

CMA-ES represents the search neighbourhood using a multivariate Gaussian distribution [18]. Gaussian adaptation takes advantage of existing candidate solutions by selecting nearby values more frequently (to be added to the population), but still allows values farther away to be explored. CMA-ES uses a scaling factor and covariance matrix to determine the size and shape of the neighbourhood distribution. By using multiple dimensions of variance, it is possible to fit the distribution more closely to the fitness landscape.

Algorithm 1 describes how we optimise the vector of model input parameters θ, from an initial mean μ_θ and standard deviation σ. The parameters are all represented as doubles (optimised within the bounds given in Section 6), but following the advice of Hansen [19], each element of θ is scaled to the range $[0, 10]$ (with $\mu_\theta = 5$ and $\sigma = 2$). At each iteration, λ new parameter sets are generated and run n times on each implementation. The covariance matrix cov is automatically updated (along with μ_θ and σ), but is initially set to the identity matrix as there is no prior information to take into account. Time series Y are generated from model implementations M1 and M2, then characterised using peak value pv, peak time pt and area under the curve auc.

4. CASE STUDY MODELS

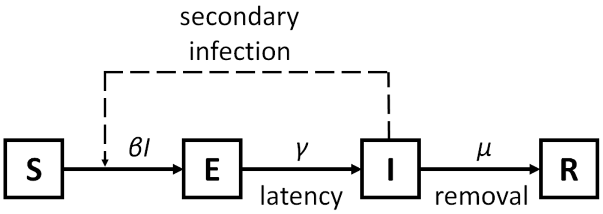
Stochastic epidemiological models are often used to help researchers understand epidemics (e.g. by estimating the total number of cases that will result from an epidemic). In this section we describe the two epidemiological models that are used in this paper as case studies for our technique.

4.1 The SEIR Model

The *SEIR* model has been used to make predictions about the spread of a wide variety of diseases (including Measles [25] and Ebola [14]). It tracks the number of hosts (e.g. people) through several exclusive compartments: *Susceptible* (not infected), *Exposed* (infected but not infectious), *Infectious* and *Removed* (dead or recovered).

Within the basic structure of an *SEIR* model, there are many options: the model may be continuous or discrete (in time and/or host in each compartment); it may incorporate explicit spatial/contact heterogeneity or assume the law of mass action. The version featured in our experiments is non-spatial, assumes homogeneous contact of hosts (i.e., mass action) and has discrete host numbers and continuous time. A graphical description of the model is shown in Figure 1.

Figure 1: SEIR model schematic

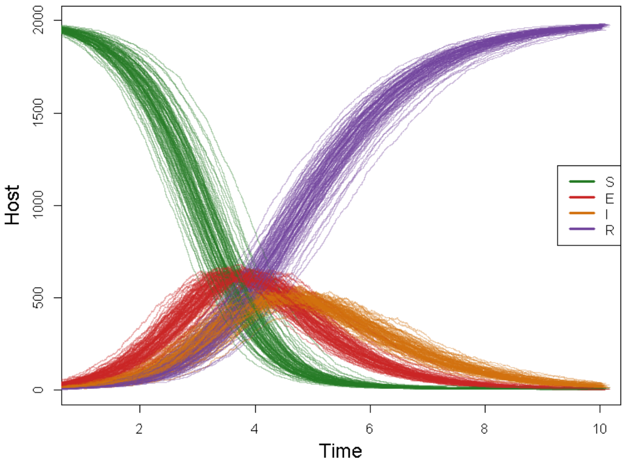


In the notation used for chemical reactions (hosts in compartments on the left of the arrow transition to hosts in compartments on the right, at the rate given above the arrow), host transitions between compartments occur as follows:

$$\begin{aligned} I + S &\xrightarrow{\beta} I + E \\ E &\xrightarrow{\gamma} I \\ I &\xrightarrow{\mu} R \end{aligned} \tag{2}$$

β is the infection rate, γ is the rate at which Exposed hosts become Infectious, and μ is the rate at which Infectious hosts are Removed. We start each simulation with 10 Infected hosts and 1990 Susceptible hosts, and simulate the epidemic progression using the stochastic Gillespie algorithm [12] with Binomial-tau leap approximation [13]. An example of the time series produced by the simulation is shown in Figure 2.

Figure 2: Example SEIR Time Series



To evaluate our technique against potential implementation discrepancies, we created a simple implementation of this model [39]. We choose values for the parameters, then modify those parameter values to simulate new implementations. Specifically, we choose a value for β, run the implementation with this value, then slightly increase β, run the implementation again, and so on. We do the same for γ.

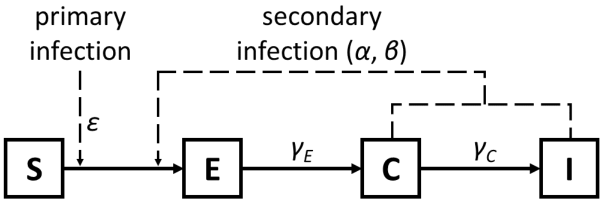
4.2 The SECI Model

In addition to artificial discrepancies in the *SEIR* model, we also apply our technique to explore differences between two real implementations of a theoretical *SECI* model used for predicting the spread of Huánglóngbìng (HLB), also known as citrus greening. HLB is caused by bacteria of the genus *Candidatus Liberibacter* and spread by psyllids [3]. It is characterised by stunted tree growth and misshaped bitter fruit, that stay green underneath. HLB is considered the most destructive citrus disease in the world, leading to reductions in yield between 30% and 100% [15].

It is important to understand how these implementations differ, as if used incorrectly, their predictions might give conflicting information. We therefore illustrate how our technique can be used to help someone, using these implementations for the first time, understand the assumptions and outputs of each implementation so that they can run the desired simulations and interpret the outputs correctly.

Figure 3 shows a graphical description of the *SECI* model. It includes compartments for: *Susceptible* (not infected), *Exposed* (infected but neither infectious nor showing symptoms), *Cryptic* (infectious but not showing symptoms) and *Infectious* (infectious and showing symptoms) hosts. When hosts die they should be transferred to the *Removed* compartment. However, since we are modelling the disease over a short period of time (and all hosts start in the Susceptible compartment), they never reach this stage. Therefore, this compartment is not explicitly included in the model.

Figure 3: SECI model schematic



The rates at which hosts move between compartments in the model depend upon the choice of parameters (α, β, ϵ, γ_E and γ_C). In this study, we fix the values of γ_E and γ_C, and adjust α, β, ϵ by our search-based optimisation technique.

Movement between the compartments is conceptually similar to that for the *SEIR* model, but more complicated because this HLB model is spatially explicit (and for that reason we omit reaction equations for this model). When there are no Cryptic or Infectious trees, the amount of time before a given Susceptible tree moves to the Exposed compartment is exponentially distributed with rate parameter ϵ. This process reflects infectious material coming from outside the citrus grove (brought by people or by wind), and is therefore termed 'primary infection'. The times spent in the Exposed and Cryptic compartments are also exponentially distributed, with rate parameters γ_E and γ_C respectively. The number of Cryptic and Infectious trees increases the rate at which Susceptible trees become infected (through 'secondary infection'). The rate at which a Susceptible host i becomes infected at time t when there are some number of Cryptic and/or Infectious trees is given in Equation 3.

$$\phi_i(t) = \epsilon + \beta \sum_j k(\frac{r_{ji}}{\alpha}) \tag{3}$$

The probability HLB will spread over a certain distance r_{ji} is given by the 'dispersal kernel' k, where $k(u) = \exp(-u)$. r_{ji} is the distance between a Cryptic or Infectious tree j and a Susceptible tree i. Each distance is scaled by the α parameter, then the kernel evaluations are summed up and multiplied by β, the rate of secondary infection. Since the summation is restricted to trees j that are Cryptic or Infectious, when there are no Cryptic or Infectious trees, this equation reduces to the primary infection term ϵ.

We compare two implementations of this HLB disease model. The context of this comparison was that a scientist was learning how to use the two implementations, and seeking to produce consistent outputs so as to be confident of using the implementations correctly. For that reason, the differences identified were due to the scientist using the implementations incorrectly, or due to 'wrapper' code that the scientist wrote to compare the outputs. The implementations were one developed by Parry *et al.* [38] (which, along with the wrapper code, we label M1) and one developed by Cunniffe *et al.* [7] (which, along with the wrapper code, we label M2). *We strongly emphasise that none of the findings of this study affect in any way the results or conclusions of those papers, or indicated that there were any faults in the original code.* M1's parameters were estimated using data from a commercial citrus plantation in Florida, USA, which had been surveyed for disease between 2005 and 2007. M2 is presented in a more general study of optimal culling strategies to combat plant disease[1]. M1 is implemented in C++ and Python, and M2 in C and Python. Program code was obtained from the respective authors. We made minor changes to M1 to make it theoretically identical to M2[2]. Simulations were started with 2000 Susceptible hosts, spatially distributed to represent part of a commercial plantation in Florida (part of the dataset used by M1 [38]).

5. RESEARCH QUESTIONS

This section presents our research questions concerning the effectiveness of the search-based pseudo-oracle technique presented in this paper; Section 7 provides our answers.

RQ1: Can the time series comparison metric reveal small discrepancies between implementations?
Before we apply search-based optimisation to maximise the differences between the implementations under test, it is important to ensure we are providing the optimisation technique with information about the discrepancies to learn from and build upon. To determine the size of the smallest error our technique can distinguish, we need to control the discrepancies introduced.

We do this by adding an increasing amount of error to the β and γ parameter values of the *SEIR* implementation, to simulate implementations with varying sizes of discrepancy (keeping the input parameters and characterising statistics the same). Kolmogorov-Smirnov tests are applied multiple times to compare the distribution of outputs from each 'implementation'. We should expect that as the discrepancies get larger, the p-values will become lower. This indicates that, when used as an objective function, the p-values should help guide the optimisation towards larger differences.

RQ2: Is the search-based technique more effective at finding differences than random testing?
It is common practice when evaluating search-based optimisation techniques, to assess whether they perform better than a random approach. This allows us to determine if it is able to utilise patterns of fitness in the input domain, rather than just aimlessly exploring the landscape. To answer this research question, we run the search-based pseudo-oracle technique on the *SEIR* implementation with varying β and γ alongside random testing, to evaluate which approach is able to achieve a higher level of fitness (i.e. lower p-values) more quickly.

RQ3: Does the search-based technique find discrepancies between real implementations of a model?
Our technique can only be considered effective if it helps us identify and understand the differences between real implementations of a model. We therefore investigate the effectiveness of our technique by applying it to two implementations of the SECI HLB model.

There will always be some degree of difference in the output, due to stochasticity, so we need to determine whether the differences found by our technique are genuine or just due to randomness. We therefore investigate them further by running the parameters selected by the technique more times. If the differences we find are consistent with our previous results, we conclude this does indicate a genuine discrepancy.

We inspect the output of each implementation and consider how the differences in their time series might have been caused. Each time we discover that the implementations are being used incorrectly, we make corrections as appropriate and then re-run the same input parameters multiple times to determine whether or not the difference has gone away.

6. EXPERIMENTAL METHODOLOGY

Experiments were conducted using two models: the simple *SEIR* model (to which we introduced discrepancies artificially) and the more advanced *SECI* HLB model (for which we compared two real implementations used to inform policy and research, as modified with wrapper code by a scientist learning how to use the two implementations). To each of these models, we applied search-based optimisation to minimise the p-values for the Kolmogorov-Smirnov tests of various statistics (AUC, peak time, peak hosts and total hosts). In other words, we maximised the differences between the distribution of time series for each implementation.

We used our technique to identify the artificially introduced differences in the *SEIR* implementation, first for the parameters (where time has units T and S, E, I and R have units $[S]$) $\beta = 0.003\ [S]^{-1}T^{-1}$, $\gamma = 1\ T^{-1}$ and $\mu = 1\ T^{-1}$, then by evolving new parameter values within the range [0,0.006] for β, [0,2] for γ and [0,2] for μ. In both this and the *SECI* HLB experiments, we allowed our search-based optimisation technique 30 generations in each trial and we ran 50 trials in total. This number of generations was chosen because we found on average that each trial took approximately one hour on the desktop computer we used to perform the experiments (Intel Core i7, 8GB of memory). We consider this a reasonable amount of time for a computational modeller to wait for the results.

We identified differences between M1 and M2, the two implementations of the *SECI* HLB model (along with wrapper code) using a similar approach. Each test case (set of input parameters) is run 100 times to address stochasticity. Test cases are generated within the plausible ranges [0,20] m for α, [0,0.1] yr^{-1} for ϵ, and [0,1000] m^2yr^{-1} for β' (which we had originally considered identical to β with units yr^{-1}, but see Section 7.3 for the distinction). We then used the results of our tests to identify differences in the time series output of the two implementations (M1 and M2). This information was used as a guide to help us inspect the code for the reasons behind these differences in output behaviour.

7. RESULTS

7.1 Answer to RQ1

RQ1 is evaluated by artificially introducing discrepancies into the SEIR implementation through slight increases to β or γ. The aim is to discern whether our technique can identify small differences and gain insight into the size and type

[1] M2 [7] contains a case study of HLB, but with different parameters. To ensure the models are the same, we use the parameters given in M1 [38].

[2] There are several options in M1 for the amount of time a host spends in the Exposed compartment; for consistency with M2, we set the amount of time to be distributed according to an exponential distribution.

of differences that can be detected. (Note in this section we are not performing any optimisation). We collect statistics from 100 simulations for each increase on each parameter and record the distribution of minimum p-values produced by the Kolmogorov-Smirnov tests (see Figure 4 and Figure 5). It is also possible to count the false positives/negatives by introducing a threshold (e.g. $p < 0.05$), but since this is somewhat arbitrary, we plot the distribution of p-values. Larger increases produce lower and less variable p-values (as can be compared with the original model at 0% increase). This indicates our technique can detect differences and provides information that may be used by the search.

Changes to β and γ affect the SEIR implementation in different ways. Increasing β raises the rate of infection, whereas increasing γ raises the rate at which hosts become infectious. The changes in the curves produced by the increased values of β and γ are subtle, but our technique can still detect them, as it uses a variety of metrics. Table 7.3 shows that beta discrepancies were found using Peak Value and Peak Time metrics, whereas gamma discrepancies could only be found using the AUC of E. If only one metric was used, discrepancies might be overlooked, but by combining multiple metrics, our technique can be made highly effective.

Many of the differences are difficult for a human observer to spot, especially when the size of error in β or γ is small. To illustrate the qualitative nature of the changes in a discernible way, Figure 6 shows the changes produced by a 4% increase in β and Figure 7 shows a 2% increase in γ. These increases were chosen for illustration, as they consistently resulted in low p-values - see Figure 4 and Figure 5).

Increasing β makes host move faster from S into E, changing the peak of E and reducing the AUC of S. Differences can be seen in Figure 6 for the peak value of E and R, peak time of E, and AUC of S and R. Since the epidemic is faster, this also affects I, but is less noticeable from the figure. Our technique is able to detect these differences, as can be seen from Table 7.3, which shows the median p-values that were recorded for each statistic. Discrepancies in γ are more difficult to observe (see Figure 5). Increasing γ makes host move faster from E into I. This increases the AUC of E, but due to the stochasticity of the model it is challenging to spot this. Nevertheless, our technique can detect the difference and Table 7.3 clearly shows a low p-value for this statistic.

The p-values for larger increases indicate a difference on every simulation, but as difference gets smaller the chance of a high p-value occurring increases. It is possible that very small differences may be overlooked, or that p-values will occasionally be low even when there is no difference. Consider for example that increases of 0.25% on β (see Figure 4) or 0.1% on γ (see Figure 5) have essentially the same distribution of p-values as the original model. It is helpful to consider a range of statistics (as we have done), since discrepancies can have various effects. By taking the minimum of these p-values, we are always looking at the greatest difference. However, when the discrepancy is very small, it is necessary to search for input values that make differences in the output more apparent. This helps to ensure the correct identification of differences between implementations.

7.2 Answer to RQ2

We evaluate RQ2 by comparing the progress of evolutionary optimisation (CMA-ES) with random search. Both approaches are applied to find input parameter values that make the differences produced (when increasing γ by 0.1%) more obvious, starting from the initial values ($\beta = 0.003$, $\gamma = 1$ and $\mu = 1$) used in the previous question. The lower bounds of each parameter in the search are 0; the upper bounds are 0.006 for β, and 2 for γ and μ. Figure 8 indicates the search-based technique to be slightly faster at achieving low p-values for increases in γ than random search. However, both random search and CMA-ES achieve low p-values within a reasonably short space of time (less than one hour).

Evolutionary optimisation could allow the search to be terminated earlier and the differences still be identified (see Figure 8). This might make the technique more attractive to scientists as it would not take as long to test their models. Another appealing aspect of CMA-ES is that since it maintains a multivariate distribution of useful parameter values during the search it is less likely the values chosen at the end will have had a low p-value due to random chance (this is a definite danger, since even when we had not introduced any difference, some low p-values were encountered). The evolutionary optimisation approach therefore has the potential to lead to faster and more consistent testing.

Figure 9 shows the distribution of p-values for 100 simulations using the parameter values chosen by CMA-ES and random search. Although the range of p-values produced for random search is slightly wider than CMA-ES, the difference is small when compared with the distribution for the original parameter values. Similarly, Figure 10 shows the CMA-ES is not always faster. When optimising parameter values for the SECI HLB model, it achieved a low p-value around the same time as random search. This suggests either approach would be suitable for use with our technique.

7.3 Answer to RQ3

RQ1 is evaluated by applying the search-based pseudo-oracle technique to the outputs of two real implementations along with wrapper code (M1 and M2) of a model used to make policy decisions for HLB, as used by a scientist learning to use those two implementations. *Once again, we emphasise the differences between M1 and M2 are* not *faults in the original code (and in no way affect the results or conclusions of [38] or [7]), but are rather differences relating to model settings, output formats, and conversion code written by the scientist learning how to use the two implementations.*

We ran our search-based technique 50 times and investigated the reasons for the differences we observed in the output of M1 and M2. Each time we found the cause of a difference, we modified the code to make M1 and M2 give the same result. In many cases, we believe the differences we found were linked to the same cause because when we changed the code these differences disappeared. Details are provided of five unique differences we identified between M1 and M2 (illustrated in Figure 11), which we believe to have been the cause of the differences in output we observed. The parameter values used to find the differences are shown in Table 2 and the p-values of the Kolmogorov-Smirnov test on 100 time series for each difference are shown in Table 3.

1) *Scaling factor.* The first difference the scientist identified when using our technique was that epidemics simulated by M2 can progress much more quickly than those simulated by M1. For example, Figure 11b shows the Cryptic compartment reaches its peak earlier in M2 than in M1. Our technique found very low p-values for many of the statistics (see Table 3), indicating that the difference was large.

Figure 4: Introducing Discrepancies into β

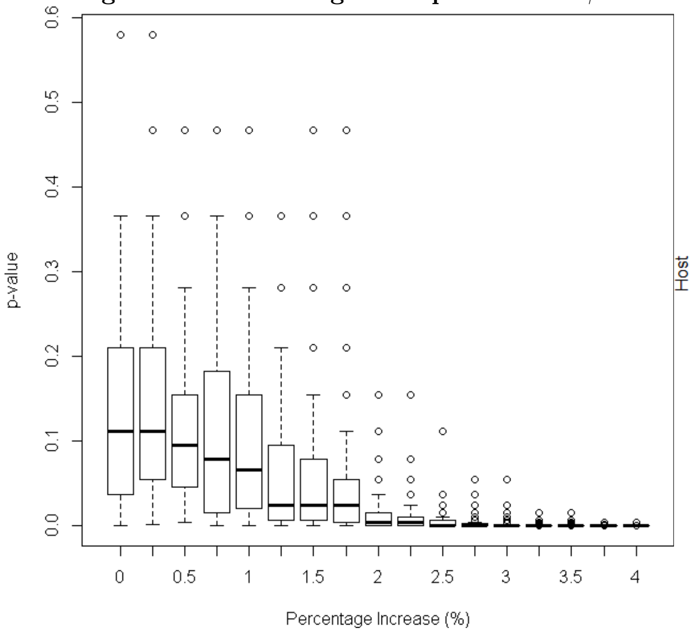


Figure 6: Identifying Discrepancies in β

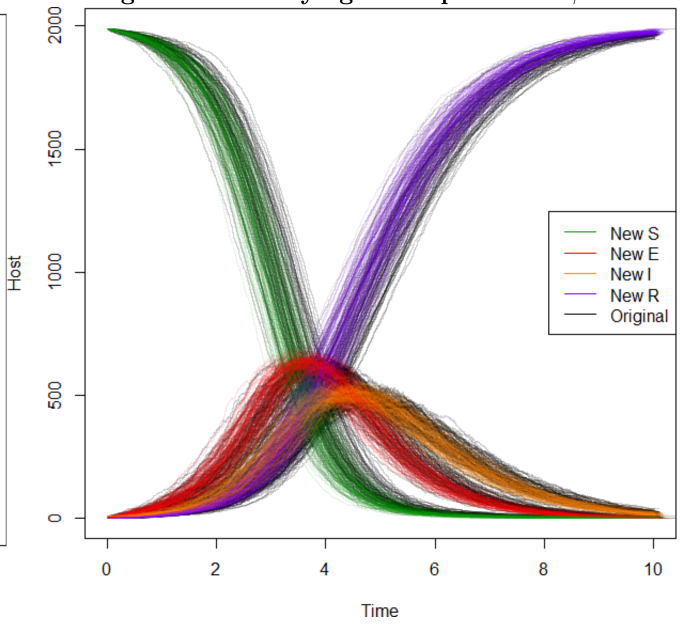


Figure 5: Introducing Discrepancies into γ

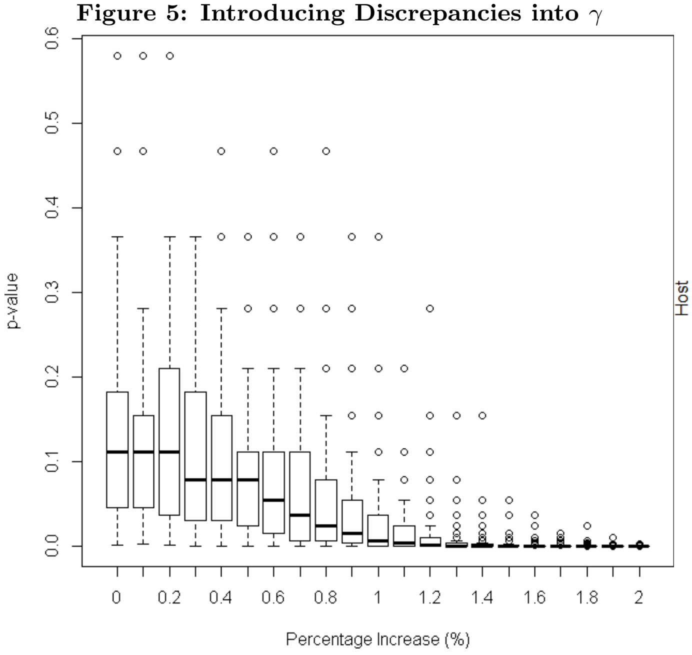


Figure 7: Identifying Discrepancies in γ

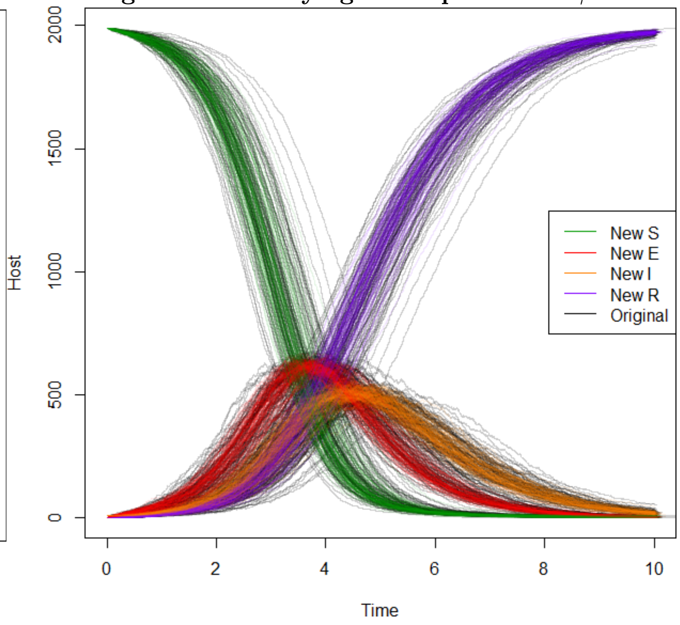


Table 1: Median p-values for 100 simulations with 4% increase in β and 2% increase in γ

Parameter	Peak Value				Peak Time				AUC			
	S	E	I	R	S	E	I	R	S	E	I	R
β	1.000	0.000	0.008	0.001	1.000	0.010	0.016	0.211	0.003	0.581	0.417	0.006
γ	1.000	0.211	0.211	0.211	1.000	0.367	0.417	0.468	0.367	0.000	0.581	0.281

Figure 8: Progress of Evolutionary Optimisation and Random Search on the SEIR Model

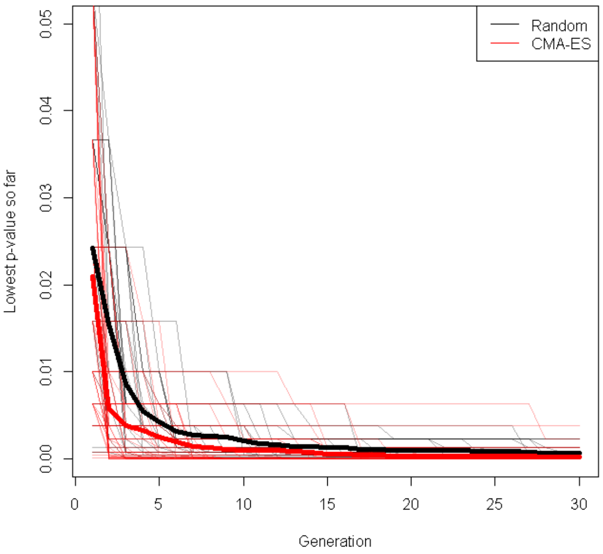


Figure 9: p-values reached at end of optimisation

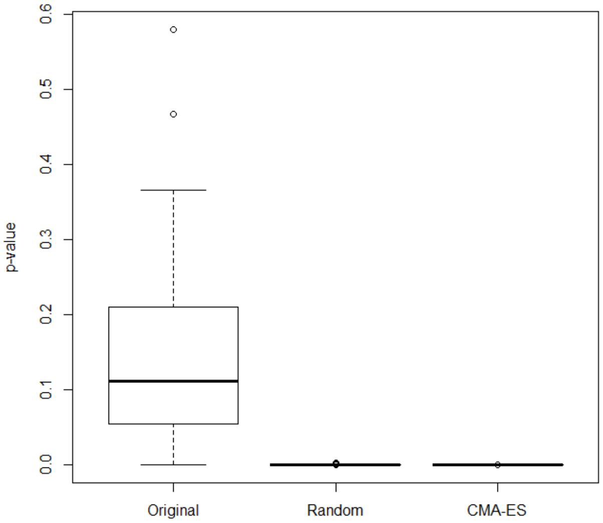
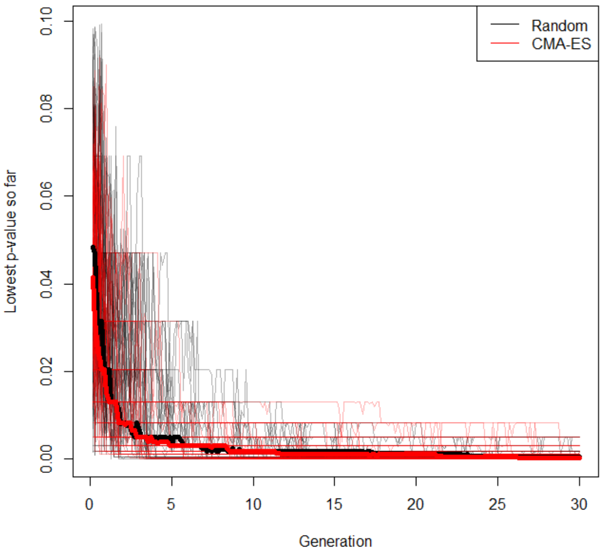


Figure 10: Progress of Evolutionary Optimisation and Random Search on the SECI HLB Model



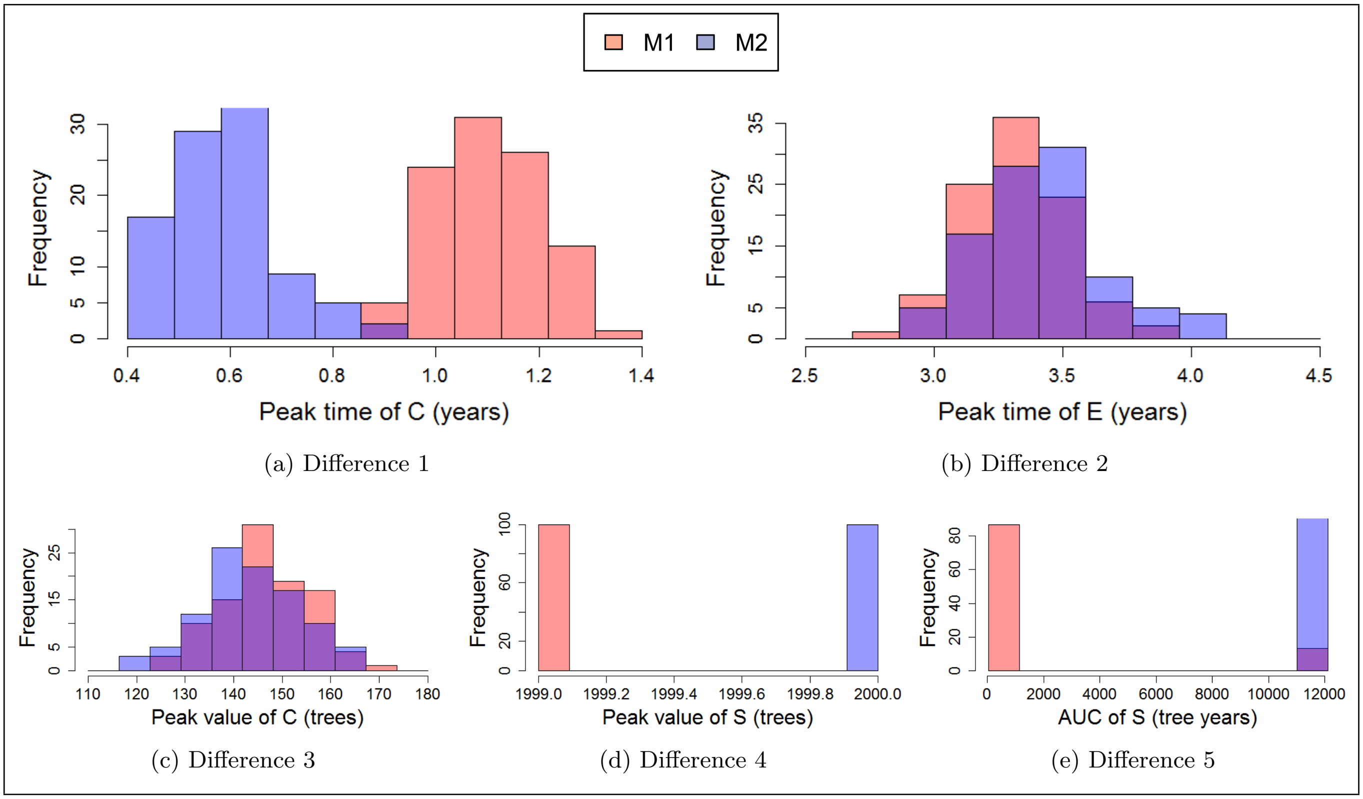
On inspection of the code, the scientist learnt that in M1 the input parameter β' is adjusted by a scaling factor $1/\alpha^2$, so that the model parameter (see Equation 3) is $\beta = \beta'/\alpha^2$. To make the two implementations have the same behaviour, we changed the wrapper code of M2 so that it performed the same conversion.

2) *Start time/age category.* After making this change, our technique found cases in which epidemics simulated by M1 progressed slightly faster than those simulated by M2. For example, Figure 11b shows the Exposed compartment reaches its peak earlier for M1 than M2. The scientist used this information to learn that the rate at which hosts move from the Exposed compartment to the Cryptic compartment depends on their age (i.e. γ_E is different). Age categories do not exist in M2, so M1 and M2 had different values for γ_E until trees were old enough to change category.

The time at which the epidemic starts is given by t_0; before t_0, ϵ is effectively 0, so none of the trees become infected. The scientist had presumed the starting time was not relevant for his purposes, so set $t_0 = 0$. However, as well as controlling the starting time, t_0 also affects the age category, thus impacting the rate at which HLB spreads because younger trees move from Exposed to Cryptic more quickly. This difference was picked up by the Kolmogorov-Smirnov test (see Table 3) and we resolved the issue by setting $t_0 = 1$.

3) *Distribution of time spent in Cryptic compartment.* In M2, the time spent in the Cryptic compartment is exponentially distributed (as is common for disease models), but in M1 it was gamma distributed. Although this was described in [38], it was overlooked by the scientist using the model. This means that even when M1 and M2 had identical means, the distributions were different, so the time series were not the same (see Figure 11c). The most obvious difference is that in M1, the Cryptic curve rises further ahead of the Infectious curve than in M2. Our technique made this difference more prominent by maximising β' (see Table 2). The scientist removed this difference by modifying the code of M1 to use an exponential distribution.

4) *Initial number of Susceptibles.* All the simulations were started with 2000 Susceptible hosts, so the peak of the Susceptible curve should not change. However, the Kolmogorov-Smirnov test (see Table 3) showed this value was different between M1 and M2. Upon closer inspection, we found the initial number of Susceptible hosts in M1 was 1999 (see Figure 11d). This was caused by a mistake in the mechanism we created to make the output formats of M1 and M2 the same. The state *after* the first transition from Susceptible



(a) Difference 1

(b) Difference 2

(c) Difference 3

(d) Difference 4

(e) Difference 5

Figure 11: Differences identified between M1 and M2

to Exposed was output to times *prior* to that transition.

5) *Modified output/truncation.* For extremely low values of ϵ (see Table 2), the rate at which hosts become infected is so small that there are simulations in which no infections occur. The wrapper code for M1 output a single-line output comprising only the initial state (with all hosts Susceptible). It was expected to contain the state at subsequent times as well, but these were not produced. This meant that the area under the Susceptible curve was often very low (see Figure 11e). The difference was found by the Kolmogorov-Smirnov test (see Table 3) in the Susceptible AUC statistic.

Table 2: Model Parameters used to Find Differences

Parameter	Difference				
	1	*2*	*3*	*4*	*5*
α (m)	11.645	11.801	18.087	7.270	2.139
β' (m^2 yr^{-1})	475.685	31.744	998.182	449.415	179.576
ϵ (yr^{-1})	0.091	0.013	0.090	0.086	5.124×10^{-6}

8. THREATS TO VALIDITY

In our experiments, we compare time series using a particular set of statistics (AUC, peak value, peak time) and statistical test (Kolmogorov-Smirnov). It is possible other statistics and tests may perform better. However, we deliberately chose the Kolmogorov-Smirnov test as it is widely used to compare probability distributions and the statistics we have chosen can detect large range of discrepancies (as is demonstrated by its detection of increases to β and γ in the SEIR implementation). The technique we have presented in this paper can be applied to a variety of different time series and it is straightforward to incorporate different statistics or tests should the user choose to do so.

Table 3: *p*-Values used to Find Differences

Statistic	Difference				
	1	*2*	*3*	*4*	*5*
AUC_S	0.000	0.005	0.556	0.961	0.000
AUC_E	0.556	0.099	0.794	0.193	0.344
AUC_C	0.047	0.000	0.000	0.443	1.000
AUC_I	0.000	0.005	0.794	0.155	1.000
PT_S	1.000	1.000	1.000	1.000	1.000
PT_E	0.000	0.047	0.992	0.894	0.344
PT_C	0.000	0.069	0.000	0.677	1.000
PT_I	0.000	1.000	0.443	0.296	1.000
PV_S	1.000	1.000	1.000	0.000	1.000
PV_E	0.000	0.894	0.556	0.961	0.344
PV_C	0.000	0.994	0.000	0.677	1.000
PV_I	0.000	0.003	0.261	0.717	1.000

Since the software under test is stochastic, the outputs may be different each time it is run. For a given sample size, there is a chance the outputs from a faulty implementation will follow the same distribution as those from the correct implementation (we could fail to find a fault) or the outputs from two correct implementations may have different distributions (we may incorrectly presume there is a fault). We have addressed this issue by incorporating a search algorithm into our technique, to identify parameter values that consistently reveal a difference if one exists. In addition, we recommend the user to run their model with the parameter values multiple times, once the search is complete, to record the distribution of p-values (in our experiments we ran the implementations 100 times). If the resulting p-values are consistently low, this suggests there is a difference, which can then be checked by an inspection of the program code.

Our search technique optimises the minimum p-value produced using the statistics. The benefit of this approach is that it ensures that at least one of the resulting differences is as large as possible. Another way to aggregate the statistics is to minimise the average p-value. However, this may lead to parameter values that alter all the time series by such a small amount that the difference is not visible. In our preliminary experiments, we also found that this more vulnerable to stochasticity than our chosen approach.

There is a danger that movement around the parameter space could be dominated by noise, even when search-based optimisation is used. This may explain why in our experiments the CMA-ES did not perform much more efficiently than random search. However, the p-values that were recorded at the end of each search (using either technique) were consistently low and we were able to find real differences between implementations of an important model. Therefore, even when the model we are testing is highly stochastic, we are still able to find parameter values that make the differences between implementations apparent.

Finally, in the SEIR example, we saw that small changes to β and γ did not necessarily produce changes in the distribution of p-values (consider the 0.5% increase to β or the 0.2% increase to γ). We might worry that this could lead to a false sense of security when the discrepancy between two implementations is very small. Yet, even in these cases, our technique was able to use search to find input parameters that consistently reduced the p-values to a low level that made the differences clear. Obviously there is a limit to the size of discrepancy our technique can detect, even when using search, as the precision of a computer is not infinite. However, our technique was even able to detect differences that were so small to be unobservable to a human.

9. RELATED WORK

Search-based software testing is a highly popular research area [33]. A large number of search-based techniques have been applied to generate test cases that achieve control flow coverage [30, 43, 26, 10]. This is the first attempt, however, at applying search-based optimisation to the challenges involved in testing implementations of stochastic models.

Search-based techniques have been used previously to identify differences between multiple versions of the same program. For example, McMinn [32] applied a genetic algorithm to maximise the difference in output between artificially produced pseudo-oracles. However, most search-based testing techniques assume the output can be compared precisely. A tool [24] has been developed to compare the relative error produced by syntactic mutants using a threshold, but as of yet it has not been applied to a search-based technique.

There has been very little research into testing stochastic software. Murphy *et al.* [36] used metamorphic relations (descriptions of properties that should remain true in every execution) to test machine learning algorithms, but did not address to potential for these relations to occasionally fail due to stochasticity. Yoo [47] also applied metamorphic testing to machine learning, but used the Mann-Whitney U test to assess whether the output was different under stochasticity. Whereas Yoo tested a program that output Boolean values [47], the implementations we tested output complex time series; Yoo assessed significance for a particular confidence interval, but we also used the p-values of each of our Kolmogorov-Smirnov tests to drive the optimisation.

10. CONCLUSIONS

We have presented a novel technique for testing stochastic software, using search-based optimisation and pseudo-oracles. Optimisation is conducted to find input parameters for which the output distributions of the implementations differ as much as possible, so the causes of these differences can be determined. We identify differences and direct the search by applying Kolmogorov-Smirnov tests to sets of summary statistics, derived from the time series outputs. Compared with simpler approaches, such as tolerance thresholds or measuring the difference in means, our technique is more robust and can identify subtle differences in the output distributions. We illustrate our technique on a simple non-spatial model and describe how it helped a scientist become familiar with the correct use of two implementations for a more complex spatial model of a real disease (HLB).

11. FURTHER WORK

In this paper, we chose to use a Kolmogorov-Smirnov test to compare the distributions of output from each implementation, but this is not the only method of comparison. The Anderson-Darling test [1] gives more weight to the tails of the distribution and there is some evidence it is more sensitive to small differences than the Kolmogorov-Smirnov test [9]. It would be worthwhile to compare these techniques along with other approaches (especially in cases with cyclical or bifurcated time series), to discover if they allow us to maximise differences between outputs more efficiently. There are also many other optimisation algorithms (genetic algorithms, stochastic hill climbing etc.) we could apply to this problem, some may be better suited than others [44].

Our technique identifies differences in implementations of stochastic models, but does not tell us where to look in the code. Automated debugging techniques, such as slicing [41] and statistical debugging [28] record which statements are exercised each time an error occurs to predict where the fault may be. Unfortunately it is not possible to use these techniques in the model implementations we tested, as their control flow is fairly linear, i.e. the same statements are run whenever the implementation is used. It may be possible instead to predict likely locations in code based on a measure of their contribution to the final output value.

12. ACKNOWLEDGMENTS

This work was supported by the University of Cambridge/ Wellcome Trust Junior Interdisciplinary Fellowship “Making scientific software easier to understand, test and communicate through modern advances in software engineering”. APC is funded by a Bill & Melinda Gates Foundation grant “Epidemiological Modelling to Inform Strategies for: (i) Detection, Management and Inoculum Reduction of Wheat Stem Rust: (ii) Monitoring and Management of Current and Emerging Cassava Virus Strains”. We thank Richard Stutt and Matt Castle for useful discussions.

13. REFERENCES

[1] T. W. Anderson and D. A. Darling. A test of goodness of fit. *Journal of the American Statistical Association*, 49(268):765–769, 1954.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.

[3] J. M. Bové. Huanglongbing: a destructive, newly-emerging, century-old disease of citrus. *Journal of Plant Pathology*, 88:7–37, 2006.

[4] F. Brayton, A. Levin, R. Tryon, and J. C. Williams. The evolution of macro models at the federal reserve board. In *Carnegie Rochester Conference Series on Public Policy*, pages 43–81, 1997.

[5] A. Carzaniga, A. Goffi, A. Gorla, A. Mattavelli, and M. Pezzé. Cross-checking oracles from intrinsic software redundancy. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 2014.

[6] N. J. Cunniffe, R. C. Cobb, R. K. Meentemeyer, D. M. Rizzo, and C. A. Gilligan. Modeling when, where, and how to manage a forest epidemic, motivated by sudden oak death in California. *Proceedings of the National Academy of Sciences (PNAS)*, 2016.

[7] N. J. Cunniffe, R. O. J. H. Stutt, R. E. DeSimone, T. R. Gottwald, and C. A. Gilligan. Optimising and communicating options for the control of invasive plant disease when there is epidemiological uncertainty. *PLoS Computational Biology*, 2015.

[8] B. Daniel, D. Dig, K. Garcia, and D. Marinov. Automated testing of refactoring engines. In *Proceedings of the 6th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 185–194, 2007.

[9] S. Engmann and D. Cosuineau. Comparing distributions: The two-sample Anderson-Darling test as an alternative to the Kolmogorov-Smirnoff test. *Journal of Applied Quantitative Methods*, 6(3):1–17, 2011.

[10] G. Fraser and A. Arcuri. 1600 faults in 100 projects: Automatically finding faults while achieving high coverage with EvoSuite. *Empirical Software Engineering*, 20(3):611–639, 2013.

[11] A. Geller and S. J. Alam. A socio-political and -cultural model of the war in Afghanistan. *International Studies Review*, 12(1):8–30, 2010.

[12] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[13] D. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Physical Chemistry*, 115(4):1716–1733, 2001.

[14] M. F. C. Gomes, A. Pastore y Piontti, L. Rossi, D. Chao, I. Longini, M. E. Halloran, and A. Vespignani. Assessing the international spreading risk associated with the 2014 West African Ebola outbreak. *PLoS Currents Outbreaks*, 2014.

[15] T. R. Gottwald. Current epidemiological understanding of citrus huanglongbing. *Annual Review of Phytopathology*, 48:119–139, 2010.

[16] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson. How do scientists develop and use scientific software? In *ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE)*, 2009.

[17] N. Hansen. cma 1.1.06: Python package index. Retrieved May 9, 2016, from Python: https://pypi.python.org/pypi/cma.

[18] N. Hansen. The CMA evolution strategy: a comparing review. In J. A. Lozano, P. L. naga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation (studies in fuzziness and soft computing)*, pages 75–102. Berlin, Germany: Springer, 2006.

[19] N. Hansen. CMA-ES source code, 2011. Retrieved May 9, 2016, from Laboratoire de Recherche en Informatique: https://www.lri.fr/~hansen/cmaes_inmatlab.html.

[20] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posik. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the 12th Genetic Evolutionary Computation Conference (GECCO)*, pages 1689–1696, 2010.

[21] P. R. Harper and A. K. Shahani. Modelling for the planning and management of bed capacities in hospitals. *Journal of the Operational Research Society*, 53(1):11–18, 2006.

[22] L. Hatton and A. Roberts. How accurate is scientific software? *IEEE Transactions on Software Engineering*, 20(10):785–797, 1994.

[23] S. Hettrick, M. Antonioletti, L. Carr, N. C. Hong, S. Crouch, D. D. Roure, I. Emsley, C. Goble, A. Hay, D. Inupakutika, M. Jackson, A. Nenadic, T. Parkinson, M. I. Parsons, A. Pawlik, G. Peru, A. Proeme, J. Robinson, and S. Sufi. UK research software survey 2014. Retrieved May 9, 2016, from Zenodo: https://zenodo.org/record/14809.

[24] D. Hook and D. Kelly. Mutation sensitivity testing. *Computing in Science & Engineering*, 11(6):40–47, 2009.

[25] M. J. Keeling and B. T. Grenfell. Understanding the persistence of measles: reconciling theory, simulation and observation. *Royal Society Proceedings B*, 269(1489):335–343, 2002.

[26] K. Lakhotia, M. Harman, and H. Gross. AUSTIN: an open source tool for search based software testing of c programs. *Information and Software Technology*, 55(1):112–125, 2013.

[27] N. Li, J. Hwang, and T. Xie. Multiple-implementation testing for XACML implementations. In *Proceedings of the Workshop on Testing, Analysis and Verification of Web Software (TAV-WEB)*, pages 27–33, 2008.

[28] C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff. Statistical debugging: A hypothesis testing-based approach. *IEEE Transactions on Software Engineering*, 32(10):831–858, 2006.

[29] F. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[30] G. McGraw, C. Michael, and M. Schatz. Generating software test data by evolution. *IEEE Transactions on Software Engineering*, 27(12):1085–1110, 2001.

[31] W. M. McKeeman. Differential testing for software.

Digital Technical Journal, 10(1):100–107, 1998.

[32] P. McMinn. Search-based failure discovery using testability transformations to generate pseudo-oracles. In *Proceedings of the 11th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1689–1696, 2009.

[33] P. McMinn. Search-based software testing: past, present and future. In *Proceedings of the 4th International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, pages 153–163, 2011.

[34] Z. Merali. Computational science: error, why scientific programming does not compute. *Nature*, 467(7317), 2010.

[35] H. Motulsky. Comparing dose-response or kinetic curves with GraphPad Prism. *HMS Beagle: The BioMedNet Magazine*, 34, 1998.

[36] C. Murphy, G. Kaiser, and L. Hu. Properties of machine learning applications for use in metamorphic testing. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 867–872, 2008.

[37] D. Orchard and A. Rice. A computational science agenda for programming language research. In *International Conference on Computational Science (ICCS)*, pages 713–727, 2014.

[38] M. Parry, G. J. Gibson, S. Parnell, T. R. Gottwald, M. S. Irey, T. C. Gast, and C. A. Gilligan. Bayesian inference for an emerging arboreal epidemic in the presence of control. *Proceedings of the National Academy of Sciences (PNAS)*, 111(17):6258–6262, 2014.

[39] M. Patrick. seir_model, 2016. Retrieved May 9, 2016, from GitHub: https://github.com/mattimpat/seir_model.

[40] J. A. Sokolowski and C. M. Banks. *Modeling and simulation fundamentals: theoretical underpinnings and practical domains.* Wiley, Hoboken, NJ, 4th edition, 2010.

[41] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3:121–189, 1995.

[42] H. Trautmann, O. Mersmann, and D. Arnu. Package 'cmaes'. https://cran.r-project.org/web/packages/cmaes/cmaes.pdf.

[43] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841–854, 2001.

[44] T. Weise. Global optimization algorithms - theory and application, 2009. Retrieved May 9, 2016, from Dr Thomas Weise: http://www.it-weise.de/projects/book.pdf.

[45] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[46] Y. Xiong, D. Hao, L. Zhang, T. Zhu, M. Zhu, and T. Lan. Inner oracles: input-specific assertions on internal states. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2015.

[47] S. Yoo. Metamorphic testing of stochastic optimisation. In *Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, pages 192–201, 2010.